Writing Web 2.0 Applications for Science Archives

William Roby,
IPAC, Caltech, Pasadena, CA USA

ABSTRACT

Writing these sorts of science archive web applications is now possible because of some significant breakthroughs in web technology over the last four years. The Web browser is no longer a glorified batch processing terminal, but an interactive environment that allows the user to have a similar experience as one might expect with an installed desktop application.

Taking advantage of this technology requires a significant amount of UI design and advanced interactions with the web server. There are new levels of sophistication required to effectively develop this sort of web application.

The IRSA group (NASA/IPAC Infrared Science Archive) is developing web-based software that equally takes advantage of modern technology and is designed to be reused easily. This way we can add new missions and data sets without a large programming effort while keeping the advanced interface.

We can now provide true web-based FITS viewing, data overlays, and interaction without any plugins. Our tabular display allows us to filter, sort, and interact with large amounts data in ways that take advantage of the browser's power.

This talk will show how we can us AJAX technology, the Google Web Toolkit (GWT), and Java to develop a data archive that is both well designed and creates a truly interactive experience.

Keywords: AJAX, Archives, GWT, Web, Web 2.0, UI, Java, FITS

1. INTRODUCTION

In the past five years, it has become very clear that the Web and web-development are revolutionizing. We are seeing applications come to the web that we never imagined before. Many web applications were once thought to be limited to the desktop. Other web applications are using levels of networking never considered before. The Web is certainly becoming more central in peoples' lives. The Web browser is the key application to have on a computer. This Web 2.0 technology provides developers with a great opportunity to create Astronomy archive websites that allow users increasingly to interact with and understand the data that they are searching for.

However, writing applications with this technology requires a lot of experience with UI design and development. We are no longer just creating a Web document, we are writing Web applications. Many are as powerful and advanced as desktop applications. At IRSA (NASA/IPAC Infrared Science Archive), we are now developing a configurable interface that will allow us to intake new dataset and front the data with a very advanced user interface. We are using XML to define the queries and how the results should be displayed and examined.

2. HISTORY AND TECHNOLOGY

2.1 A Very Brief History of Computing

To really understand this change, it is helpful to ask the question, "How did Microsoft make it big?" The answer is found in how business worked before the PC. Everything was batch. The user used a batch mainframe terminal. He filled out a form, hit enter, and waited for the results. Microsoft brought interactive programming into a mainframe world. The way business used computers changed significantly.

Software and Cyberinfrastructure for Astronomy, edited by Nicole M. Radziwill, Alan Bridger,
Proc. of SPIE Vol. 7740, 774010 · © 2010 SPIE · CCC code: 0277-786X/10/$18 · doi: 10.1117/12.857794

The Web in the '90's and early 2000's was very similar to this mainframe batch model. The user filled out a form, hit submit, and waited for the results; except this time he did it with a Web browser. Computing had returned to the batch model.

This approach was shattered on Feb 8, 2005 when Google introduced Google Maps. For the first time we saw a Web browser with an interactive application. A new standard was set for web development. AJAX web development began.

Now, we are seeing AJAX-based web applications appear everywhere. Google has introduced several more applications such as Calendar, Word processor and the very popular Gmail. Yahoo has followed suit and revamped their whole web-site to be AJAX-based. They have some very powerful finance tools. The news websites are using this approach to some degree. The social media and entertainment sites are making heavy use of AJAX. There are all sorts of specialty sites such as doodle.com or rememberthemilk.com that are good examples of this new development approach. Everybody is being affected in some way. Websites that are not using AJAX elements are looking very dated.

This approach has now been dubbed Web 2.0. However, the name really just represents the groups of technologies behind the revolution.

2.2 Technologies Behind the Revolution

There are several technologies behind this Web 2.0 development revolution. The most obvious is AJAX. AJAX is an acronym (Asynchronous JavaScript And XML) that represents a set of existing components that are being used together in new ways. The major technologies are as follows:

- JavaScript
- Dynamic
- HTML
- Asynchronous Calls
- CSS
- DOM
- XML
- JASON

Lets look at some of these in more detail.

2.3 DOM

Dynamic HTML technology allows any part of a web page to change without reloading the page. The HTML document is viewed as a tree that can be modified using JavaScript. This view of the documents is called the Document Object Model (DOM). When the DOM is modified, the page changes immediately. This allows the page to change according to what the user is doing.

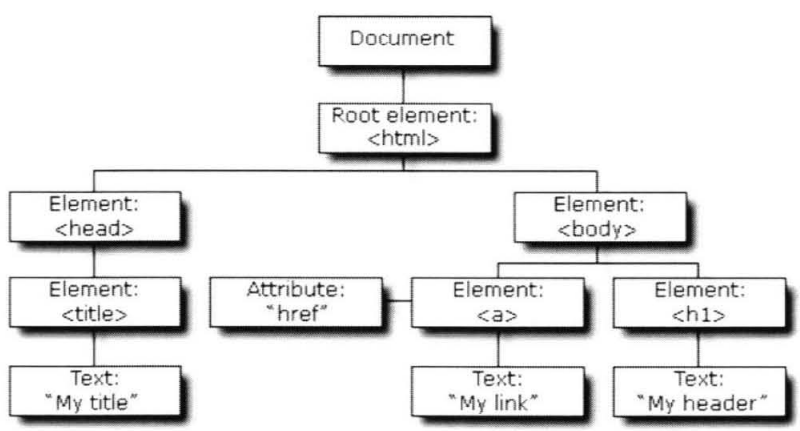

Figure 1. Document Object Model (DOM) Example

2.4 JavaScript

The maturing of JavaScript is probably the most important of the changes we are seeing. JavaScript is a very powerful, flexible, C-like scripting language that runs natively in the browser. It is event driven. This means that anytime the user

clicks or moves his mouse over or types on any part of the web page, a JavaScript function can be called. It also has access to the DOM so it can change some or all of the web page on any event.

2.5 Asynchronous Calls

Another key to JavaScript is that it can make asynchronous calls to the server. This is not the traditional page load call. JavaScript can call the server at anytime, get the results, and can change any part of the web page. This allows for updates to the web page without the user even knowing that the server is being called. The AJAX approach works with many small, quick calls to the server and updates part of the web page instead of doing large page reloads.

2.6 Differences Between AJAX and Traditional Web Pages

The following two lists show the difference between AJAX-based pages and traditional web pages

- Classis Web
  - One server call = Whole Page Load
  - Server call = UI Generation
  - Server call = Large Overhead
  - Server call required for useful results
- AJAX Web
  - One Server call = Page update
  - One Server call = data only
  - One Server call = small overhead
  - Page can generate results without server call

Figure 2 shows the flow of a traditional web application while Figure 3 shows the flow of an AJAX web application.

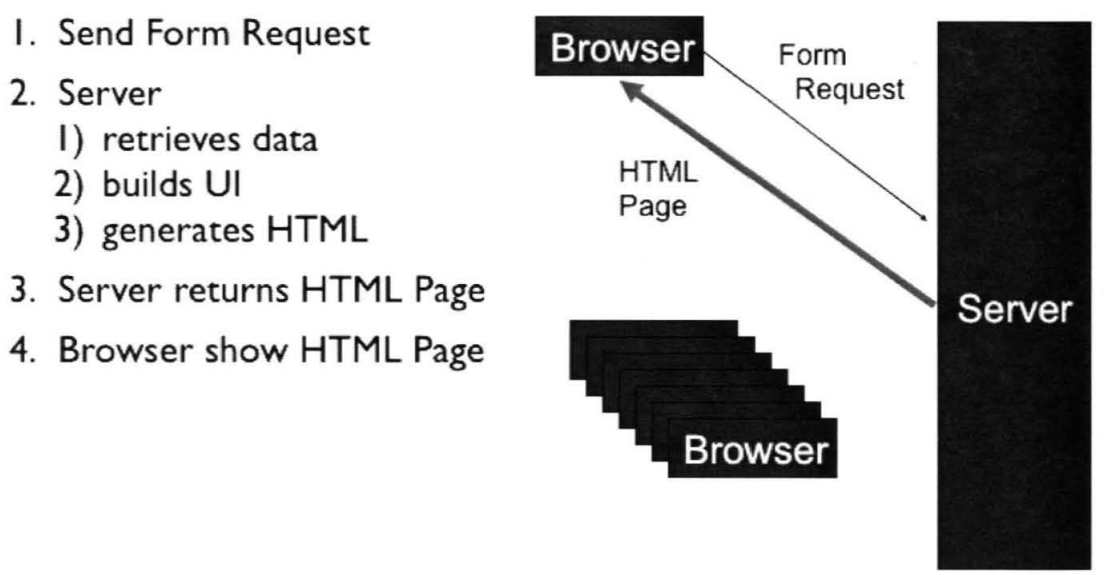

Figure 2. Traditional Web App

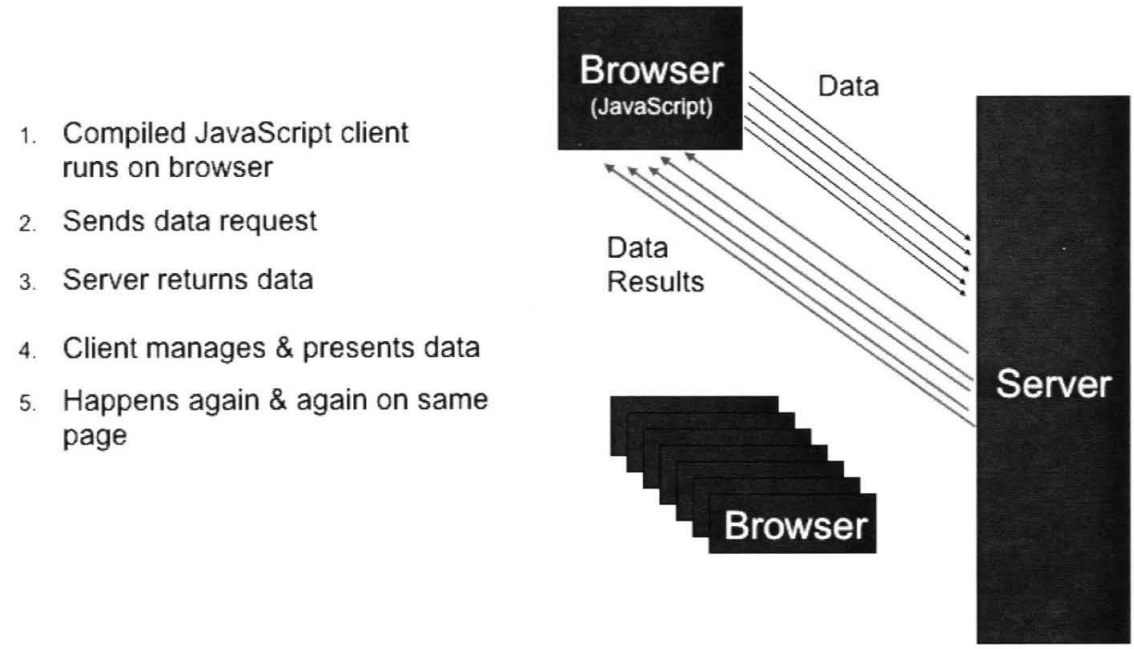

Figure 3. Web2.0 - AJAX / JavaScript Web App

3. WEB 2.0 ARCHIVE WEBSITES

3.1 Opportunities

A Web 2.0 archive website provides significant opportunity for the science community. Searching an archive for specific data can be a challenging task. In the past, archives have produced large volumes of results without giving the user tools to help decide what he really needs. We are now able to deliver more advanced web applications that can allow the user to do more analysis before he makes the decision to download potentially gigabytes of data.

We can now build tools into the Archive Web Applications that are interactive and require minimal server interaction.

Archive websites should be good at doing fast searches. They should also excel in helping the user to understand the search results. Historically the focus has been on the former. Web 2.0 capabilities now allow us to focus on the latter. Lets take a look at some capabilities we are building into our archive applications.

3.2 Smart Clients

Since the server is not required to do all the processing anymore, the user's desktop can do more work. Algorithms such as "*applying a projection to a point on an image to convert to the J2000 coordinate system*" can be done completely on the client. HTML calculation can be done on the client. While AJAX uses more calls to the server, it also offloads the server from having to do as much work. This allows for a fast server that can support more users and add more functionality.

3.3 Interactive Tables

We have written a table display that does interactive sorting, paging, and filtering. The idea here is that a user might get back 100,000 or more results in a search. Using our table, the user could sort those results on any column or further filter the results. This allows the user to do a search, then interactively explore those results. Data can be selected and shown in the FITS visualizer.

3.4 FITS Visualization

We are putting a full FITS visualizer directly into the browser. This is written without plugins. It will run in all major browsers. A user has very similar capabilities that one might have with a desktop application such of DS9 or Spot.

3.5 Spectrum Visualization

Just like we can display FITS files, we can also display spectrum interactively. The user can move his mouse over the plot and see the data. He can also zoom in or out one part of the Spectrum to see more fine grain detail.

3.6 Validating Input Forms

Traditional websites will have the user fill out a search form and then submit it. Only then will the user know that he entered some data incorrectly. Instead, we are doing validation interactively. The user gets immediate feedback if he entered an invalid value.

3.7 Putting It Together

When these types of components are assembled, then we can create some very advanced applications. This requires a well-designed infrastructure to manage these components and allow them to work together. Figure 4 is a screen show of tables and visualization working together.

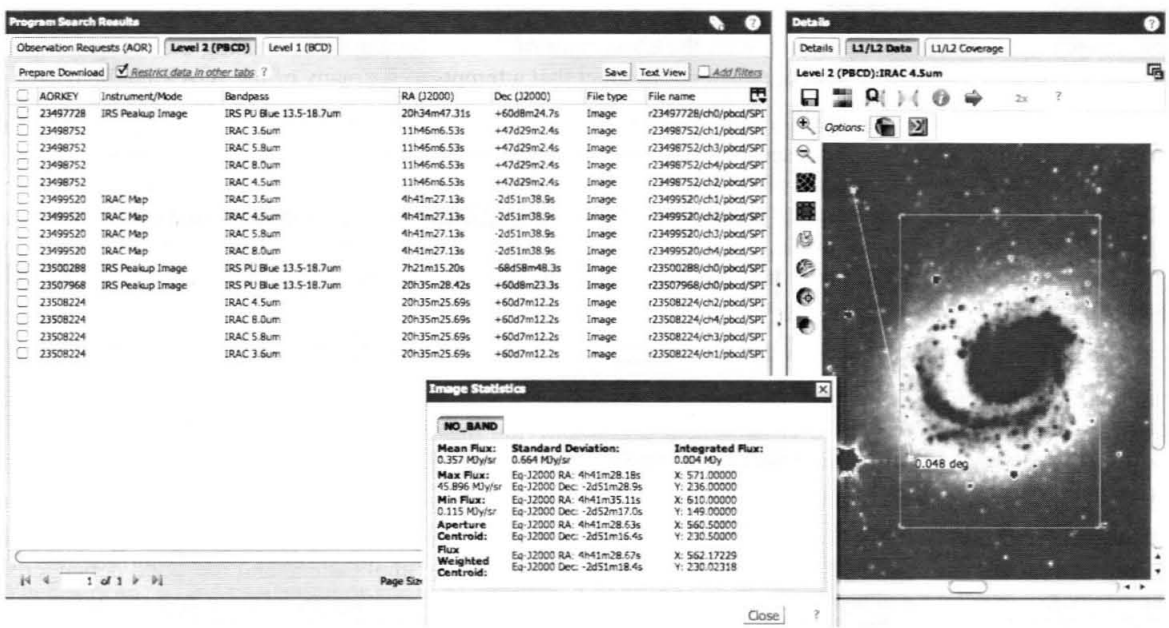

Figure 4. Three tables (one in each tab) show data while the visualizer on the right shows the details of one of the rows. When a row is clicked, a new image appears. This image is not a preprocessed JPEG, but an interactive display of a FITS file. No page reloads are required for all this to work together.

4. CHALLENGE #1: AJAX & JAVASCRIPT

4.1 Challenge

Doing AJAX development has significant challenges. Browsers can work differently. The JavaScript API can be slightly different. What will work in one browser may not work in another. There are many, many subtleties, which require constant checking of the browser type as well as code that is full of "if's" just to do simple things.

JavaScript also has some significant weaknesses. It is not strongly typed like C or Java. It has a weak debugger and development environment support. It is not a compiled language. Syntax errors are not caught until runtime. It also means that it can't be optimized until runtime. JavaScript is designed to be a scripting language. So, it has the standard scripting languages' weaknesses, such as not working well in large applications. It has particular issues when more than one programmer is working on a project.

AJAX development provides us with great opportunity. We can write stunning web applications that will be a great benefit to the user and be easy to use. It also has non-trivial obstacles. It is hard to debug and requires lots of testing on browsers. Some think that you must be an "AJAX guru" to even attempt this sort of development. It is also easy to write bad JavaScript code.

4.2 Solution - Google Web Toolkit

The best solution we have seen for dealing with the challenges of AJAX development is using the Google Web Toolkit (GWT). GWT is a well-supported, free, open-source product that attempts to fix many of the AJAX challenges.

Using GWT, the developer writes code in Java instead of JavaScript. GWT compiles the Java into JavaScript making JavaScript into sort of a web browser assembly language. This compilation creates all sorts of advantages. The biggest is that is does not just generate one set of JavaScript output. Instead, it creates a set of JavaScript output per browser type. Therefore, it can deal with most of the browser differences at compile time instead of runtime. It also compiles only what you actually use. For example, if you have a set of utility functions such as a math library, GWT will only bring in those functions that are actually used in the code. This way code is not downloaded to the browser that is not used, saving bandwidth. No browser plugin is required to make this happen.

GWT also provides several other features that are all optional to use.

It has optional UI support so you can develop in a way similar to Java Swing. You can also build the UI by accessing the DOM directly. It provides an RPC environment for simplifying making asynchronous calls to the server. It is also possible to put bits of JavaScript directly into your GWT code. GWT allows you to do anything you can do in JavaScript and provides additional features.

Using this sort of development, we see many benefits. GWT handles many cross browser issues. We get the benefits of compilation and producing more optimal code. Java works better for large applications and has good debugger support. GWT comes with a development environment and test browser that interacts with your favorite java debugger.

Figure 5 illustrates how the process of GWT development progresses. The developer starts writing in Java. Using GWT tools, the developer runs his code in a test browser. He can attach his favorite debugger to it if he desires. This iterative process continues until the developer is satisfied. When ready, the code is compiled into JavaScript, one compilation per browser type. Finally, the code is deployed onto a web server.

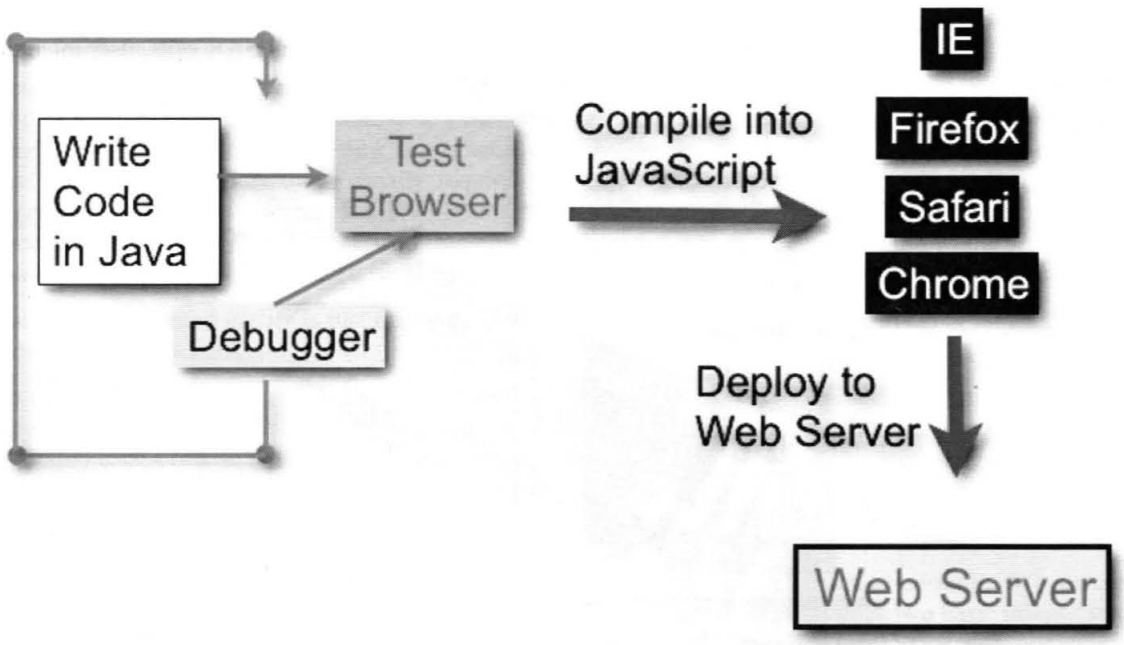

Figure 5. Development Process in GWT

5. CHALLENGE #2: FITS VISUALIZATION

5.1 Challenge

A FITS visualizer has been a desktop application. If you want to view a FITS file from a website, you must either look at a pre-stretched and colored jpeg picture or download and run your desktop FITS visualizer. Coverage plots are static. There is no real interaction with the data beyond looking at a picture.

5.2 Solution – Web-based FITS Visualizer

We have written a full web-based FITS visualizer that runs without any plug-ins. The Spitzer Spot planning tool has a fairly advanced Java/Swing based FITS visualizer as part of its common code distribution. Using GWT, we split this code up into a client piece and a server piece. The server piece can read the FITS file and create the image. The client side will maintain projection information and plot data on top of the plot. All the interactive tools exist on the client side. When it needs a new projection or zoom of the data, it calls the server. When plotting catalogs or utilities such as a distance tool, it simply does it without a server call. The code has gone through several passes of optimization, smart caching and performance tuning. Now the user can be presented with complete tools to examine his data before he ever chooses to download anything. Through several optimizations, the tool is quite fast. We find that, after a while, the user forgets he is using a website because the visualizer acts like a desktop application.

Currently, our visualizer has the following features:

- Zoom
- Grid overlay
- Crop
- Statistics tool
- Rotate North
- Distance tool
- Image Center
- Three Color Plot
- Saving the fits file or as a png
- Change Color
- Modify Stretch using multiple algorithms
- View FITS Headers
- RA/DEC/Flux readout in J2000, B1950 and Galactic

To understand how this works, let's look at the following example. As the user moves his mouse over a FITS image, (Figure 6) the information about where the mouse is on the image is updated. One field (flux) requires a server call and is only done when the mouse is paused. The following chart (Figure 7) shows the AJAX calls, user interaction, and the dataflow as the user request a FITS image and then moves his mouse.

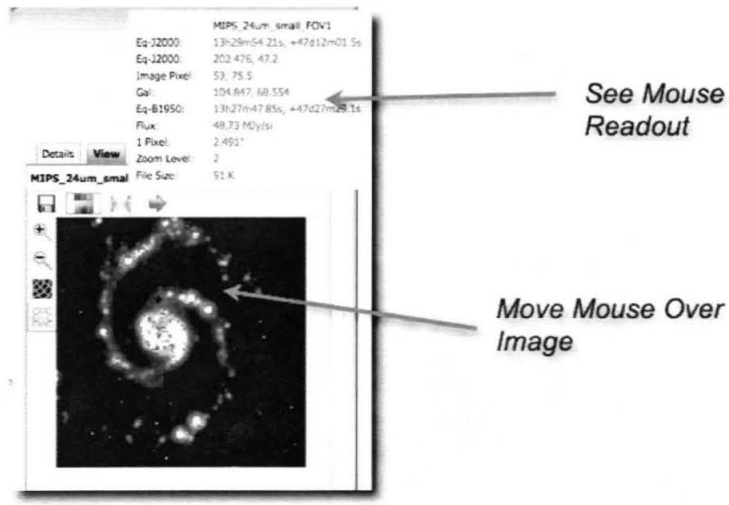

Figure 6. Example of the AJAX client / server interaction as the user moves his mouse across a FITS image

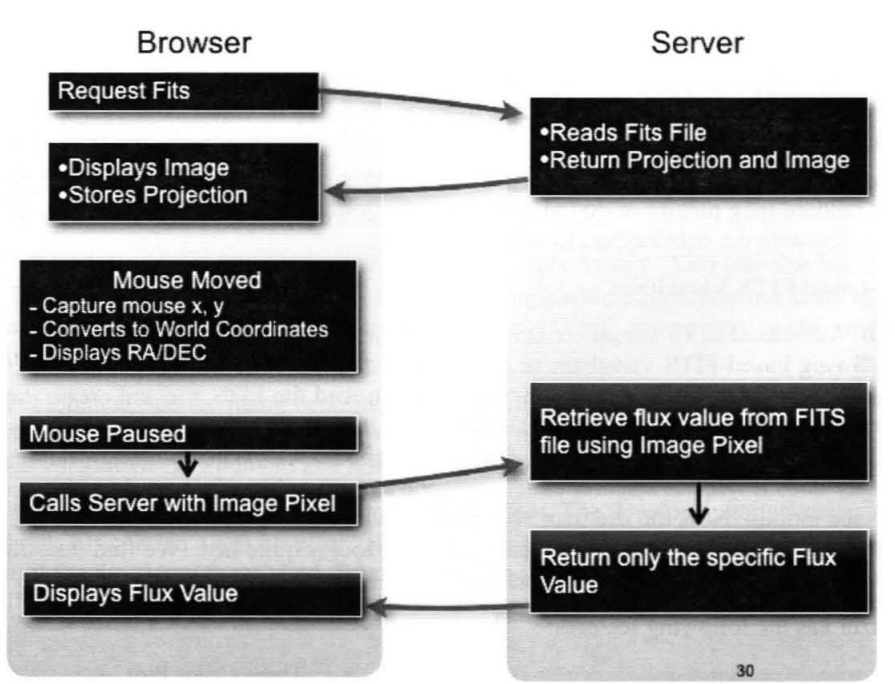

Figure 7. Example of the AJAX client / server interaction as the user moves his mouse across a FITS image

6. CHALLENGE #3: UI DEVELOPMENT IS HARD

6.1 Challenge

Web 2.0 technology moves much of the work load to an interactive JavaScript program running on the browser. As the browser becomes a smarter client it conceptually becomes very similar to Desktop UI programming. The problem is that UI development is hard. Instead of being linear and straightforward, UI programming is event-oriented and asynchronous. To create a good solution usually requires a lot of code. Use cases need to be well understood. UI components need to be thoroughly written to allow for a good user experience. User feedback, appropriate controls, tool tips, icons, and intuitive layout should all be well implemented. When this is not done well, the whole web application will appear shoddy and will often not be well liked by the user.

The FITS visualizer mentioned earlier is just one example of this complexity. Showing fast paging tables that sort and filter has many similar challenges.

At IRSA, we are regularly ingesting new datasets to serve. Each of these datasets has specific requirements. However different, they all require some form of tabular and visualization display. We need to be able to write advance applications around each of these datasets. Since the requirements are all slightly different, recreating a different UI for each would be very time consuming and resource intensive.

6.2 Solution – Configurable Archive Interface

Because writing Web apps is difficult, we are now looking for ways to reuse code as much as possible. We are creating an archive system with configurable user interfaces. The idea is that an archive Web UI does two basic things. It shows forms and displays results. If this behavior can be represented in an XML file, then we can put very powerful, reusable components into the Web applications with a minimal amount of UI development work. Now that we have developed advanced web components such as interactive tables, FITS viewers and smart form we are using an XML file to define how the search and results should look. Then we arrange the search dynamically. This will allows IRSA to ingest new data and deploy it very quickly.

We use an XML file for each web application that we want to present. The XML files contain sections for each type of query that the user might want to do against the data. This section not only defines the UI of the form for the query but also defines how the results are to be displayed and what components are to be used. Components would include such things as one or more tables to show the results, an interactive coverage plot, one or more plots of the FITS data, and possibly a spectrum. Each of these displays can change as the user clicks through the data without a page reload. Figure 8 shows the data form laid out using XML. Figure 9 shows the xml result layout.

The components can be laid out in a variety of ways with multiple instances of each component.

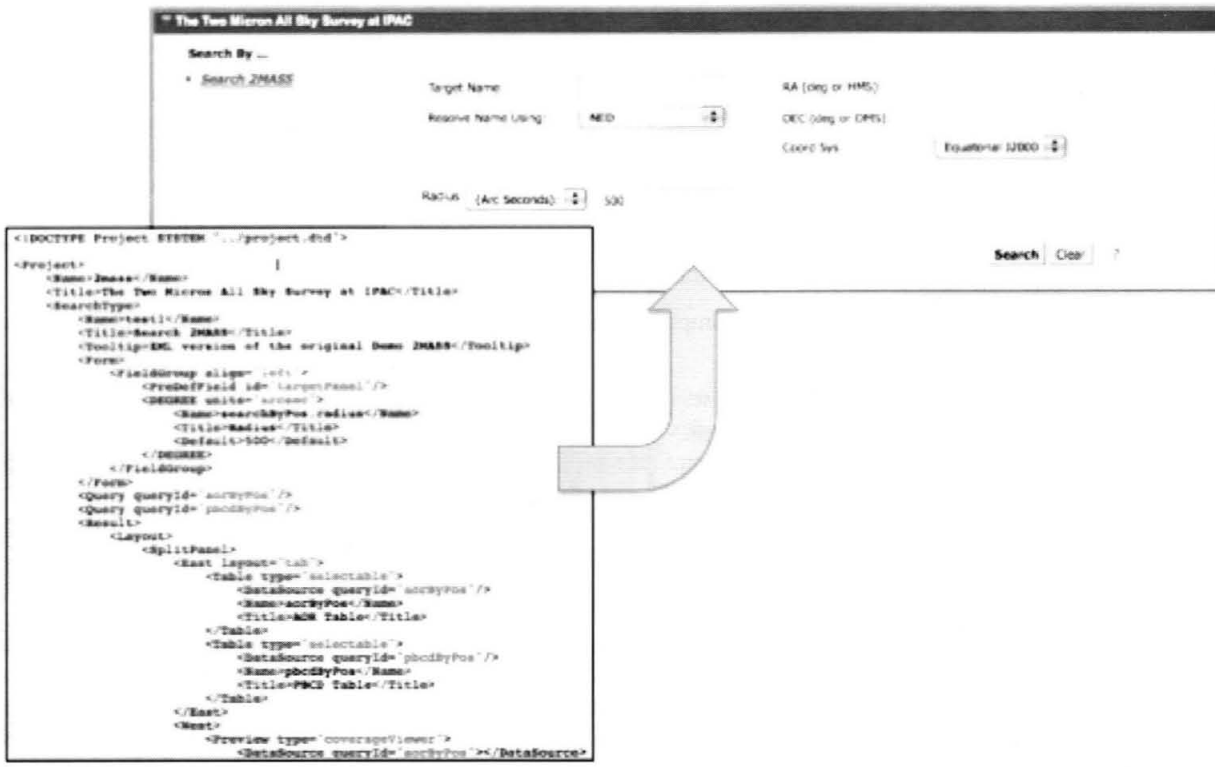

Figure 8. XML form configuration

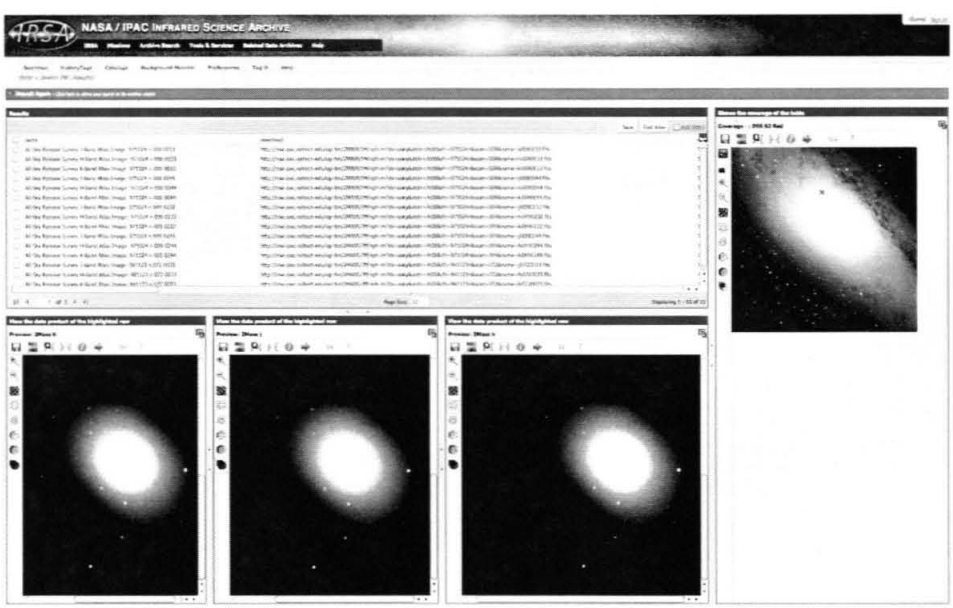

Figure 9. XML results configuration

7. CONCLUSION

We can see that, while writing AJAX web applications has its challenges, it also has the greater rewards of better quality, more power, and certainly more user-friendly applications. We have found using GWT is key to tapping into the power of this sort of development. However, doing this type of development is still time consuming if you want to do it well. Therefore, writing a reusable, configurable UI seems to be required to get the most out of our efforts. This is a significant undertaking, but it will allow quantities of different datasets to be served with minimal work.

WEBSITES

[1] Google Web Toolkit - http://code.google.com/webtoolkit/
[2] Spitzer Heritage Archive - http://shaweb1.ipac.caltech.edu/applications/Spitzer/SHA/Heritage.html
[3] Spot common distribution - https://audrey.ipac.caltech.edu/spot-common/